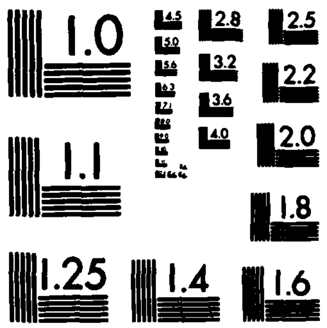
AD-A144 413 LOGLISP SEQUENTIAL FORMS WITH RESOLUTION SEMANTICS(U) 1/1
ROME AIR DEVELOPMENT CENTER GRIFFISS AFB NY R C SCHRAG
JUL 84 RADC-TM-84-13

UNCLASSIFIED F/G 9/2 NL

END


1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

(12)

RADC-TM-84-13
In-House Report
~~June~~ 1984
July

LOGLISP SEQUENTIAL FORMS WITH RESOLUTION SEMANTICS

Robert C. Schrag

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441

RADC-TM-84-13 has been reviewed and is approved for publication.

APPROVED: [illegible]

SAMUEL A. DINITTO, JR.
Chief, Command & Control Software
Technology Branch
Command and Control Division

APPROVED: [illegible]

RAYMOND P. URTZ, JR.
Technical Director
Command and Control Division

FOR THE COMMANDER: [illegible]

DONALD A. BRANTINGHAM
Acting Chief, Plans Office

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TM-84-13
6a. NAME OF PERFORMING ORGANIZATION Rome Air Development Center 6b. OFFICE SYMBOL *(If applicable)* COES	7a. NAME OF MONITORING ORGANIZATION N/A
6c. ADDRESS *(City, State and ZIP Code)* Griffiss AFB NY 13441	7b. ADDRESS *(City, State and ZIP Code)*
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL *(If applicable)* COES	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N/A
8c. ADDRESS *(City, State and ZIP Code)* Griffiss AFB NY 13441	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
62702F	5581	19	12

11. TITLE *(Include Security Classification)*
LOGLISP SEQUENTIAL FORMS WITH RESOLUTION SEMANTICS

12. PERSONAL AUTHOR(S)
Robert C. Schrag

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Yr., Mo., Day)*	15. PAGE COUNT
In-House	FROM 20Jan84 TO 2Feb84	July 1984	18

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES

FIELD	GROUP	SUB. GR.
9	2	14
9	2	15

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
LogLisp
Logic Programming

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This memorandum describes an extension to the semantics of the LogLisp artifical intelligence programming language that allows resolution in sequential forms. The execution cycle of LogLisp and the resolution semantics of existing special forms are summarized, and the need for and definitions of resolution semantics for sequential forms are presented.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER *(Include Area Code)*	22c. OFFICE SYMBOL
Robert C. Schrag	315-330-2748	RADC (COES)

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

1. Introduction.

This memorandum describes an extension to the semantics of the LogLisp artificial intelligence programming language [Robinson and Sibert 81a,b] that allows resolution in sequential forms. The execution cycle of LogLisp and the resolution semantics of existing special forms are summarized, and the need for and definitions of resolution semantics for sequential forms are presented.

2. LogLisp's Execution Cycle

LogLisp is a synthesis of Lisp and logic programming, implemented in Lisp. The logic programming component of LogLisp, called Logic, differs from conventional logic programming systems in two important ways: it uses a heuristic, non-backtracking search strategy in processing resolvent nodes; and it allows Lisp expressions to be included in knowledge base clauses and queries, providing appropriate reduction and resolution semantics for useful forms. Before resolution with a goal is attempted, that goal is first reduced, or Lisp-simplified, by performing evaluation of Lisp forms, conditioned on the instantiation of any Logic variables contained in them.

If the chosen goal is Lisp-evaluable, then that goal is considered to succeed if its value is non-NIL. It is considered to fail if the value is NIL. If simplification reduces the goal only partially or not at all, resolution is then attempted, in the following order: using facts, or ground clauses; using special resolution rules; and using rule (non-ground) clauses. Ordinary unification is used in resolving with facts and rules.

Special resolution rules exist for the following forms: =, AND, OR, and COND. The equality rule just attempts to unify its subforms. The AND special form is essentially non-operative syntactic sugar—it pushes its subforms onto the goal list of the parent node to create a new resolvent node. The OR special form pushes each of its subforms onto the parent node's goal list separately, creating a new node for each. The non-determinism of the Logic search strategy renders the Logic OR's subforms uncoupled as compared to the tight sequential dependence of Lisp's OR. In Lisp, the COND form is equivalent (modulo unit clauses) to an OR form of AND forms containing the "test" subform of the clause followed by a PROGN (implicit) of the clause's remaining subforms. This Lisp equivalence is illustrated in Figure 1.

```
          Lisp COND                      Equivalent Lisp OR, AND, PROGN

(COND (TESTA RESULTA1 RESULTA2)      (OR (AND TESTA
                                              (PROGN RESULTA1 RESULTA2))
      (TESTB RESULTB)                    (AND TESTB RESULTB)
      (TESTC))                           TESTC)
```

Figure 1

LogLisp's Execution Cycle

The COND special form of Logic exists to permit resolution on clause test subforms. Because of the standard if-then-else sequential connotation of COND, the Logic resolution semantics of COND preserves Lisp's sequential treatment of the OR in the equivalent form, rather than creating separate, independent deduction nodes for each AND subform (clause). The COND special form resolves on the test of its first clause, generating resolvent nodes which include continuations that store both the clause's "result" forms (implicit PROGN) to be processed in case the resolvent process succeeds, and the COND's remaining clauses to be processed in case all this clause's (test's) resolvent processes lead to failure. While no special resolution rule exists in current LogLisp for PROGN, it is a "feature" of the reduction semantics for a top-level PROGN that its last subform has predicate status. This feature extends as well to the implicit PROGN's of COND clause tails.

With these observations, the forms of Logic can be classified into two types: predicate forms, which occur at the top level of clauses, as AND and OR special form top-level subforms, as COND special form test subforms, and as the last subform of explicit and implicit PROGN forms with predicate status; and functor forms, the subforms at any level of non-special predicate forms. Predicate forms either succeed or fail. A Lisp predicate form succeeds when it evaluates to other than NIL; it fails if it evaluates to NIL or is unevaluable (and has no Logic definition). A Logic predicate succeeds if it can be resolved using a clause in the knowledge base or a special resolution rule, and fails otherwise. The success properties of predicate forms are summarized in Table 1.

	Logic predicate forms		Lisp predicate forms	
	resolvable	unresolvable	evaluable	unevaluable
success	always	never	if non-NIL	never
failure	never	always	if NIL	always

Table 1

Logic and Lisp functor forms differ only in that the latter can be affected by simplification before unification and the former are not.

LogLisp's Execution Cycle

Table 2 is provided to clarify the predicate and functor compositions of Lisp forms for which predicate subforms can occur.

predicate	functor
(AND . predicate-forms)	(AND . functor-forms)
(OR . predicate-forms)	(OR . functor-forms)
(PROGN . (functor-forms | predicate-form))	(PROGN . functor-forms)
{COND combines compositions for OR, AND, PROGN--see Figure 1}	

Table 2

Note that in a predicate PROGN form only the last subform has predicate status. All other (Lisp, functor) subforms are evaluated for side-effect only. The predicate PROGN form fails if any of these Lisp side-effects cannot be performed (because of uninstantiated Logic variables).

3. LogLisp and Side-effects

LogLisp combines Lisp, a side-effecting (multiple-assignment) programming facility, with the side-effect-free (single-assignment) formalism of logic programming. The logic programming variables of Logic can only be instantiated through unification, but the provision for Lisp simplification affords the opportunity for arbitrary Lisp side-effects to be performed by Logic code. Common side-effecting operations of Lisp are listed in Table 3.

effect performed by	effect accessible through
PUTPROP	GETPROP
RPLACA	CAR
RPLACD	CDR
DEFINEQ	pname
SETQ	pname {!}

Table 3

When these multiple-assignment Lisp operations are performed by Logic code, their effects can be communicated to subsequent operations in that code by virtue of the Logic/Lisp (reduction) interface. The effects of PUTPROP's and RPLAC's are available through the appropriate accessor functions. The effect of a DEFINEQ is accessible through the identifier's pname when it appears at the head of a list. Because of LogLisp's implicit quoting of non-head proper identifiers, the effect of a SETQ is available only through explicit EVALuation of the bound identifier. This poses no real problem in Logic code, and EVAL serves as a signal that reference to a global Lisp object is being made.

Communication among the predications of a Logic clause is of course also done with logic programming variables instantiated in unification, as in any other logic programming dialect. Unification is the single-assignment operation whose effects (bindings) are accessible through reference to variable identifiers, much as Lisp variable bindings are (in Lisp). In the non-terminal subforms of Logic's only present sequential form--PROGN--, however, only Lisp communication mechanisms are available. Their lack of predicate status precludes the Logic communication mechanism of unification. This poses a problem for sequential Logic computations which must make subsequent reference to intermediate results.

LogLisp and Side-effects

SETQ (and other Lisp side-effecting forms) are unattractive for this purpose because they require the creation of an object global to the entire Logic computation, to satisfy a strictly temporal need. Preceeding the PROG with a local variable declaration predicate guaranteed to succeed (as defined by the clause: (|- (Local-Logic-variables . any-Logic-variable-names))) is perhaps less disagreeable, but variable declaration is contrary to logic programming principles. A mechanism to allow resolution with the non-terminal subforms of sequential Logic forms solves this problem.

4. The New Sequential Forms

Successful programming with Logic sequential forms using communication of intermediate values requires that all subforms possess a status which permits assignment operations, including unification. The sequential form should fail if and only if the operation specified by one of its subforms cannot be performed. For a Lisp subform, failure will occur when evaluation is impossible. When the subform is fully instantiated, the evaluation's results or actual binding or not binding of (global) Lisp objects do not matter to its success. For a Logic subform, failure will occur when the subform leads to immediate or ultimate failure. The desired success properties that subforms confer on sequential forms are summarized in Table 4.

	Logic sequential subforms		Lisp sequential subforms	
	resolvable	unresolvable	evaluable	unevaluable
success	always	never	always	never
failure	never	always	never	always

Table 4

Terminal Lisp subforms in PROGN predicates represent an exception to the above success specifications, since intuitively one expects a PROGN to "return" its last value and this is the current semantics. The InterLisp implementation of LogLisp V2M3 [Schrag 83] has been modified to extend resolution semantics to PROGN, with sequential subform treatment. The function #RESPROGN is employed. Note that this resolution semantics also extends to the implicit PROGN's of COND clause tails.

A "pure" sequential form, SEQUENCE, is also provided. This form always succeeds if the side-effects of its subforms can be performed, and treats the terminal subform no differently from its predecessors. This form is also implemented in InterLisp LogLisp V2M3. SEQUENCE is defined as a Lisp function, reduction semantics for it are bestowed by #VALSEQUENCE, and resolution semantics for it by #RESSEQUENCE.

The New Sequential Forms

Figure 2 demonstrates the utility of resolution semantics for Logic sequential forms with two example clauses from a pattern-matching production rule interpreter. Logic predicates and functors are shown as identifiers with initial letter capitalized and remaining letters lower case. If input can be Matched against the clause´s specfic Pattern, then, if a SEQUENCE of global Lisp actions can be performed, the clause succeeds.

<u>equality unification</u>

```
(|- (Interpret input) <- (Match input (Patterna x y))
                         (SEQUENCE (= tempvar (CALCULATE x y))
                                   (GLOBAL-ACTION1 tempvar)
                                   (GLOBAL-ACTION2 tempvar)))
```

(a)

<u>deduction unification</u>

```
(|- (Interpret input) <- (Match input (Patternb x))
                         (SEQUENCE (Deduce tempvar x))
                                   (GLOBAL-ACTION1 tempvar)
                                   (GLOBAL-ACTION2 tempvar)))
```

(b)

Figure 2

Two types of temporary Logic variable instantiation are illustrated. In Figure 2(a) a Logic variable is instantiated to the value of a Lisp function call using the = special resolution form. In Figure 2(b) a Logic variable is instantiated by Deduction (or data base look-up) of a Logic relation.

The code for implementing functions mentioned follows. Obvious modifications to #REDINIT and #INITX, and definitions for AUTOPROGN and AUTOSEQUENCE, have also been made.

The New Sequential Forms

```
(#RESPROGN
  [LAMBDA NIL
    (PROG (HDTL LN NDRV)
          (SETQ HDTL (#TSHOW (CDR #HD)
                             #HDK))
          (COND
            ((EQ #TLL 0)
              (RETURN)))
          (SETQ LN (ADD1 LP))
          [SETQ NDRV (COND
              (*HISTORIES (COND
                            (NOHIST (CAR #HIST))
                            ((CONS (QUOTE ((PROGN . Rule)
                                           PROGN-RULE
                                           (Special Rule)))
                                   #HIST]
          (SETQ RESULT (CONS (CONSM LN NDRV #CNTN
                                    (CONS (LIST #HDK (CAR HDTL)
                                                (CONS (QUOTE PROGN)
                                                      (CDR HDTL)))
                                          #SEGTL)
                                    #C)
                             RESULT))
          (RETURN])
```

The New Sequential Forms

```
(SEQUENCE
  [NLAMBDA L
    (EVAL (CONS (QUOTE PROGN)
                L))
    T])

(#VALSEQUENCE
  [LAMBDA NIL
    (PROG (#D #RPRG #PRGL)
          (SETQ #RE (SETQ #RPRG NIL))
          (SETQ #PRGL (CDR #E))
      LOOP(#ULTX #PRGL #I)
          (COND
            ((NULL #PRGL)  **COMMENT**
              (SETQ #RE T)
              (RETURN T))
            ((#VARIABLE #PRGL)
              [COND
                ((SETQ #R #RPRG)
                  (SETQ #RE (CONS (QUOTE SEQUENCE)
                                  #PRGL]
              (RETURN NIL)))
          (SETQ #D (CAR #PRGL))
          (SETQ #PRGL (CDR #PRGL))
          [COND
            ((#VALRED #D #I)
              (SETQ #RPRG T)
              (GO LOOP))
            (#Q [COND
                  (#PRGL (SETQ #RE (CONSM (QUOTE SEQUENCE)
                                          #RE #PRGL]
                (SETQ #R T))
            [#PRGL (COND
                     ((SETQ #R #RPRG)
                       (SETQ #RE (CONSM (QUOTE SEQUENCE)
                                        #D #PRGL]
            ((SETQ #R #RPRG)
              (SETQ #RE (LIST (QUOTE SEQUENCE)
                              #D]
          (RETURN NIL])
```

The New Sequential Forms

```
(#RESSEQUENCE
  [LAMBDA NIL
    (PROG (HDTL LN NDRV)
          (SETQ HDTL (#TSHOW (CDR #HD)
                             #HDK))
          (COND
            ((EQ #TLL 0)
              (RETURN)))
          (SETQ LN (ADD1 LP))
          [SETQ NDRV (COND
              (*HISTORIES (COND
                            (NOHIST (CAR #HIST))
                            ((CONS (QUOTE ((SEQUENCE . Rule)
                                           SEQUENCE-RULE
                                           (Special Rule)))
                                   #HIST]
          (SETQ RESULT (CONS (CONSM LN NDRV #CNTN
                                   (CONS (LIST #HDK (CAR HDTL)
                                               (CONS (QUOTE SEQUENCE)
                                                     (CDR HDTL)))
                                         #SEGTL)
                                   #C)
                             RESULT))
          (RETURN])
```

5. References

[Robinson and Sibert 81a] J.A. Robinson and E.E. Sibert. The LogLisp User's Manual, unpublished interim technical report, 1981.

[Robinson and Sibert 81b] J.A. Robinson and E.E. Sibert. LogLisp Implementation Notes, unpublished interim technical report, 1981.

[Schrag 83] R.C. Schrag. Notes on the Conversion of LogLisp from Rutgers/UCI-Lisp to InterLisp, RADC-TM-83-1, 1983. AD# A127718